Identifying Vulnerable GitHub Repositories in Scientific Cyberinfrastructure: An Artificial Intelligence Approach

Sagar Samtani, Ph.D.
Assistant Professor, Grant Thornton Scholar, and CACR Fellow
Kelley School of Business, Indiana University
Joint work with B. Lazarine (IU), H. Zhu (UTSA), and H. Chen (UArizona)

*This material is based upon work supported by the National Science Foundation under Grant No. NSF OAC-1917117 (CICI)*

Presentation Overview

Project Overview: Background and Research Questions (10-12 minutes)
1. My Background
2. Cybersecurity Motivation and Existing NSF initiatives
3. Overview of GitHub cybersecurity issues
4. Research Objectives

Data Collection and Proposed Methodology (10-12 minutes)
1. Research Gaps and Questions
2. Data Collection
3. Vulnerability Assessment
4. Proposed VADW

Selected Results and Conclusion (10-12 minutes)
1. Experimental Results
2. Case Study Results
3. Conclusion
My Background

• Assistant Professor of Operations and Decision Technologies (ODT) in the Kelley School of Business at Indiana University.
  • NSF Scholarship-for-Service (SFS) fellow.
    • SFS → train students for cybersecurity positions.

• Research interests:
  • Domain – cybersecurity → cyber threat intelligence, AI for cybersecurity, scientific CI cybersecurity, Dark Web Analytics
  • Methods – AI → deep learning, network science, data/text/web mining, visualization

Why Cybersecurity?

• Cyber-attacks cost the global economy over $450B annually (Graham, 2017).
• Labelled as a national strategic priority.
• Three major NSF cybersecurity programs: SaTC, SFS, CICI
Overview of NSF CICI Program

- **NSF CICI program**: CISE OAC; 2015-present; ~50 awards; $50M+ awarded; ~120+ PIs from 50+ universities.

- Motivated by protecting (NSF) scientific cyberinfrastructure (CI), data, IP, etc.

- **Selected Topics of interest:**
  - Secure Scientific Cyberinfrastructure (SSC): encourage novel trustworthy design approaches, frameworks, and models for integrated and secure scientific CI
  - Scientific Infrastructure Vulnerability Discovery: Proactive vulnerability discovery.

Project Background and Objectives

- **Our Project** → CICI: SSC: Proactive Cyber Threat Intelligence and Comprehensive Network Monitoring for Scientific Cyberinfrastructure (CI).
  - **PI**: H. Chen (UAri zona); **Partners**: E. Skidmore (CyVerse), P. Troch (LEO).

- **Our Project’s Key Objectives:**
  1. Collect, categorize, and analyze a large-scale of hacker forum exploits.
  2. **Scan critical scientific CI assets for vulnerabilities**
  3. Automatically link exploits and vulnerabilities via deep learning-based methods
     - Prioritize links for dissemination at operational intel workshops and relevant conferences.

Focus of today’s talk.
Summary of Prevailing Scientific CI Assets

<table>
<thead>
<tr>
<th>Asset Category*</th>
<th>Sub-categories</th>
<th>Description</th>
<th>Example(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruments</td>
<td>Networked systems</td>
<td>Instruments with TCP/IP capabilities</td>
<td>Telescopes, microscopes</td>
</tr>
<tr>
<td></td>
<td>Sensor networks</td>
<td>Instruments with sensing technologies</td>
<td>Water samplers</td>
</tr>
<tr>
<td>Facilities and Hardware</td>
<td>Networked equipment</td>
<td>Devices facilitating CI communication</td>
<td>Routers, switches</td>
</tr>
<tr>
<td></td>
<td>Desktops and laptops</td>
<td>Standard systems for producing science</td>
<td>Employee workstations</td>
</tr>
<tr>
<td></td>
<td>Servers</td>
<td>Provides services to scientific CI assets</td>
<td>Databases, GPU farms</td>
</tr>
<tr>
<td></td>
<td>Physical facilities</td>
<td>Physical structures for scientific CI</td>
<td>Climate control, lighting</td>
</tr>
<tr>
<td>Systems</td>
<td>User portals and front-end</td>
<td>Designed to access back-end data assets</td>
<td>Web portal, user logins</td>
</tr>
<tr>
<td></td>
<td>Back-end and file store</td>
<td>System to store scientific CI data</td>
<td>Database, file systems</td>
</tr>
<tr>
<td>Software and Configurable Technologies</td>
<td>3rd Party or off-the-shelf</td>
<td>Open source or commercial software</td>
<td>Docker, MongoDB</td>
</tr>
<tr>
<td></td>
<td>Customized software</td>
<td>Custom software created by scientists</td>
<td>Proprietary APIs</td>
</tr>
<tr>
<td></td>
<td>Configurable VMs</td>
<td>Virtual machines that users can customize</td>
<td>Atmosphere</td>
</tr>
<tr>
<td></td>
<td>Code repositories</td>
<td>Repositories for code and data storage</td>
<td>GitHub, GitLab</td>
</tr>
<tr>
<td></td>
<td>Custom virtual networks</td>
<td>User-configurable networks</td>
<td>OpenStack</td>
</tr>
<tr>
<td>Scientific CI Data</td>
<td>Public</td>
<td>Finished or non-sensitive data</td>
<td>Finalized publications</td>
</tr>
<tr>
<td></td>
<td>Non-public</td>
<td>Data that cannot be publicly released</td>
<td>Proprietary/regulated data</td>
</tr>
<tr>
<td></td>
<td>Internal (private)</td>
<td>Data not intended to be released</td>
<td>Notes, intermediate results</td>
</tr>
<tr>
<td></td>
<td>Documentation</td>
<td>Record-keeping related to assets/processes</td>
<td>Wikis, blogs, manuals</td>
</tr>
</tbody>
</table>

Table 1. Summary of Selected Scientific CI Assets (Adapted from Peisert et al. 2020)

- **Scientific CIs are increasingly relying on:**
  1. Configurable VMs to allow users to customize their computing environments.
  2. Social coding repositories to share and distribute code for scientific reproducibility.

Research Background

- Insecure coding practices can lead to the spread of vulnerabilities that can disrupt scientific CI.

Figure 1. GitHub Repository pages include (a) name of the owner and repository, (b) the number of times the repository has been forked (copied by other users), and (c) source code in the repository.
### Summary of Vulnerability Assessment Tools

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Example(s)</th>
<th>Sample Vulnerabilities Detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>General Purpose</td>
<td>Probes conventional enterprise IT devices (e.g., workstations, servers)</td>
<td>Nessus</td>
<td>Outdated software, buffer issues, unencrypted Telnet</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Qualys</td>
<td>FTP vulnerabilities, database SQL injections</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OpenVAS</td>
<td>Weak authentications, backdoor detection</td>
</tr>
<tr>
<td>Web Application</td>
<td>Detects vulnerabilities in web technologies</td>
<td>Burp Suite</td>
<td>XSS, SQL injection, NoSQL injection, illicit parameters</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Arachi</td>
<td>DOM issues, stack traces, function signatures</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Acunetix</td>
<td>Improper input validation, out-of-bounds read, CSRF</td>
</tr>
<tr>
<td>Network Traffic</td>
<td>Monitors network traffic to detect network level vulnerabilities and threats</td>
<td>Nmap</td>
<td>Certificate injections, unauthorized commands</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Aircrack-ng</td>
<td>Replay attacks, de-authentication, fake access points</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Wireshark</td>
<td>Bandwidth issues, slow networks, packet characteristics</td>
</tr>
<tr>
<td>Code Analysis</td>
<td>Examines code structure and conventions to detect insecure coding and leaked data</td>
<td>Flawfinder</td>
<td>Weak cryptography, file permissions, insecure functions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Trufflehog</td>
<td>Secrets, passwords</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bandit</td>
<td>Deprecated libraries, insecure connections, XML attacks</td>
</tr>
</tbody>
</table>

**Table 2. Summary of Selected Vulnerability Assessment Tools**

- **Key Observations:**
  1. Prevailing scanning categories were not designed for comprehensively capturing the vulnerabilities introduced in social coding repositories.
  2. While code analysis tools can help in this regard, they can return large quantities of results.
  3. Novel approaches are needed to automatically identify, categorize, and prioritize code repositories.

### Literature Review: SCR Research

<table>
<thead>
<tr>
<th>Year</th>
<th>Author</th>
<th>Focus</th>
<th>Data Source</th>
<th>Vulnerability Assessment</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>2020</td>
<td>Lazarin et al.</td>
<td>Vulnerable Repository and User Detection</td>
<td>623 GitHub repositories and 2,401 users from one scientific CI</td>
<td>Software vulnerabilities linked to repositories</td>
<td>Unsupervised graph embedding</td>
</tr>
<tr>
<td>2019</td>
<td>Fan et al.</td>
<td>Cross Platform User Identification</td>
<td>42,840 Stack Overflow users with GitHub links in their profile</td>
<td>None</td>
<td>Attributed Heterogenous Information Network</td>
</tr>
<tr>
<td>2019</td>
<td>Meli et al.</td>
<td>Data Leakage in Public GitHub Repositories</td>
<td>681,784 repositories from GitHub API 3,374,973 from Google GitHub Snapshot</td>
<td>API key visibility at a repository level</td>
<td>Regular expressions</td>
</tr>
<tr>
<td>2018</td>
<td>Bana and Arora</td>
<td>Identify key SCRs and users</td>
<td>8,000 GitHub users</td>
<td>None</td>
<td>Bipartite network analysis</td>
</tr>
<tr>
<td>2018</td>
<td>Meng et al.</td>
<td>Secure coding challenges</td>
<td>503 Stack Overflow posts</td>
<td>Code snippet level</td>
<td>Manual classification</td>
</tr>
<tr>
<td>2017</td>
<td>Yang et al.</td>
<td>Use of Stack Overflow in GitHub</td>
<td>909,000 GitHub repositories</td>
<td>None</td>
<td>Similarity analysis</td>
</tr>
</tbody>
</table>

**Table 3. Selected Recent Social Coding Literature Identifying Vulnerabilities in Source Code from GitHub**

- **Key Observations:**
  1. Past studies have focused on general SCR analysis; little work has explored multiple scientific CIs.
  2. There is a lack of comprehensive vulnerability scanning for all source code in SCRs.
  3. Given the relationship of users and repositories, the prevailing methods are network science-based.
     - However, past studies have not included vulnerability assessment into their analysis.
Research Objectives

• The scale of scientific CIs and their repositories necessitates an automated approach to assess vulnerabilities and group similar repositories for prioritization.

• Grouping repositories can be achieved by leveraging recent advances in deep learning (DL) and graph embedding methods (Samtani et al., 2020).

• In this work, we aim to automatically:
  • Detect the vulnerabilities in scientific CI GitHub repos with code vulnerability assessment tools.
  • Group repositories with similar vulnerabilities (based on severity and quantity) to facilitate vulnerability prioritization and remediation efforts.

Presentation Overview

Project Overview: Background and Research Questions (10-12 minutes)
1. My Background
2. Cybersecurity Motivation and Existing NSF initiatives
3. Overview of GitHub cybersecurity issues
4. Research Objectives

Data Collection and Proposed Methodology (10-12 minutes)
1. Research Gaps and Questions
2. Data Collection
3. Vulnerability Assessment
4. Proposed VADW

Selected Results and Conclusion (10-12 minutes)
1. Experimental Results
2. Case Study Results
3. Conclusion
Research Questions and Design

- We propose the following questions for study:
  - How can we identify vulnerabilities within GitHub repositories?
  - How can we group repositories based on their vulnerabilities and relationships?

![Figure 2. Proposed Research Design](image)

GitHub Data Collection

Scientific CI 1

Scientific CI 2

Vulnerability Assessment

- Bandit
- Flawfinder
- Flaw Finder
- Gitrob
- Trufflehog

Graph Construction and Projection

$G=(U, R, E, F)$ Bipartite Networks

Proposed VADW

Vulnerability Severity Feature Weighting

Experiments and Case Study

- Experiment #1: Cluster Quality for All Repository Types
- Experiment #2: Cluster Quality for Root Repositories Only
- Case Study: Clustering CI 1’s GitHub Ecosystem

Research Design: GitHub Data Collection

- We collected repository (name, owner, files) and user (name, organization, repositories) data from two large-scale and long-standing NSF-funded CIs.
  - **CI 1**: founded in 2008; $115M total funding from NSF; 70,000 life-science users from approximately 5,000 academic and 2,400 non-academic institutions.
  - **CI 2**: Founded in 1956; collaborates with 115 colleges and universities.

- With the GitHub API, we collected every repository hosted by each organization and their forked repositories.
Research Design: GitHub Data Collection

- 2,793 repositories from 4,889 distinct users; 389,707 commits and 1,931 forks for all repositories across both CIs.

- CI 2 has significantly more repositories, forking activity, and contributions to forked repositories than CI 1. The top two languages are C++ and Python.

<table>
<thead>
<tr>
<th>Type</th>
<th>Sub-category</th>
<th>Vulnerability</th>
<th>Description</th>
<th>Example</th>
<th>Bandit</th>
<th>Flaw Finder</th>
<th>Gitrob</th>
<th>Trufflehog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secret</td>
<td>Secret</td>
<td>Secret</td>
<td>A potential password/key</td>
<td>73b569ec227ef1a132d6923aab21463271cd</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Password</td>
<td>Word password found</td>
<td></td>
<td></td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Cryptography</td>
<td>Weak crypto.</td>
<td>Insufficient crypto. Method v.size = cryp.key.size();</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Filetype</td>
<td>Filetype</td>
<td>Filetype</td>
<td>File that may contain secrets</td>
<td>Django configuration file</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Permission</td>
<td>File permission</td>
<td>File permission</td>
<td>File may have dang, Permissions file.err.code = chmod(filePath, 0664);</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Insecure method</td>
<td>Insecure function</td>
<td>Function can be vulnerable</td>
<td>Use of insecure and deprecated function (mktemp).</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Insecure module</td>
<td>Module can be vulnerable</td>
<td>Pickle can be unsafe when used to deserialize untrusted data</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Deprecated library</td>
<td>Library no longer supported</td>
<td>The pyCrypto library is no longer actively maintained</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Internet</td>
<td>Insecure conn</td>
<td>Dangerous internet connections</td>
<td>Requests call with verify=False disabling SSL certificate checks.</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Injection</td>
<td>Insecure input</td>
<td>Dangerous handling of user input</td>
<td>Use of unsafe yamldump. Allows instantiation of arbitrary objects.</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Attack</td>
<td>SQL injection</td>
<td>Hardcoded SQL expressions</td>
<td>Possible SQL injection vector through string-based query construction.</td>
<td></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>XSS</td>
<td>XSS attack</td>
<td>Dangerous XML library</td>
<td>Using xmlrpc to parse untrusted XML data is vulnerable.</td>
<td></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>XSS vulnerability</td>
<td>Dangerous library usage</td>
<td>By default, xmlrpc sets autoscapec to False.</td>
<td></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 4. GitHub Data Collection

Research Design: Vulnerability Assessment

- We selected four open source vulnerability assessment scanners based on language coverage, usability, and age (Table 5):
  - Bandit: scans for 11 secrets, insecurities, and attacks in Python (Torkura et al., 2016).
  - Flaw Finder: scans for four secrets, insecurities, and attacks in C (Kaur et al., 2020).
  - Gitrob: scans for three secrets and insecurities in GitHub (Roy et al., 2017).
  - Trufflehog: scans for two secrets in GitHub (Meli et al., 2019).

- We assessed the vulnerabilities of all repositories in both CIs with these tools.
Research Design: Vulnerability Assessment

<table>
<thead>
<tr>
<th>Dataset</th>
<th># of Repositories</th>
<th>Vulnerabilities</th>
<th>Vulnerability Distribution</th>
<th>Top Vulnerable Repository</th>
<th>Number of Vulnerabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>CI 1</td>
<td>328</td>
<td>Secrets 9,036</td>
<td>13.06%</td>
<td>8,613</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>High 1,825</td>
<td>2.64%</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Medium 2,377</td>
<td>3.44%</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Low 55,951</td>
<td>80.87%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CI 2</td>
<td>2,170</td>
<td>Secrets 24,025</td>
<td>1.70%</td>
<td>39,244</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>High 50,896</td>
<td>3.60%</td>
<td>36,860</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Medium 65,220</td>
<td>4.61%</td>
<td>36,850</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Low 1,274,979</td>
<td>90.10%</td>
<td>36,844</td>
<td></td>
</tr>
</tbody>
</table>

Table 6. Summary of GitHub Vulnerability Scan for Scientific CIs

- **Key Observations:**
  1. CI 2 has the highest number of vulnerabilities at 1,415,120.
  2. Both scientific CIs have a similar vulnerability severity distributions with at least 80% of vulnerabilities being low severity.
  3. The four CI 1 repositories with the highest number of vulnerabilities are all forked.
     - Illustrates the significance of forked repositories to the propagation and preservation of vulnerabilities in a GitHub ecosystem.

Research Design: Graph Construction and Projection

- GitHub follows a bipartite network structure that can be projected into two monopartite networks to identify key users and repositories (Wang et al., 2020; Lazarine et al. 2020).

- Therefore, we denote the bipartite network as $G=(U, R, E, F)$, where:
  - $G$ is a directed graph.
  - $U$ is the node set, $\{u_1, u_2, u_3, \ldots, u_n\}$, of all users that have contributed to a repository.
  - $R$ is the node set, $\{r_1, r_2, r_3, \ldots, r_n\}$, of all repositories.
  - $E$ is the edge set, $\{e_1, e_2, e_3, \ldots, e_n\}$, of directed edges from a user contributing to a repository.
  - $F$ is the feature matrix of each node; number of vulnerabilities from each user or repository.

- We project the bipartite network into two monopartite networks (one for repositories, one for users).

- Given the focus of this study on grouping vulnerable repositories, we leverage only the repository monopartite network for the subsequent analysis.
Graph Embedding and Evaluation

- Grouping users and repos based on their relationships and vulns. without apriori knowledge requires an unsupervised graph embedding method that accounts for nodal features and operates on directed graphs (Table 13).

<table>
<thead>
<tr>
<th>Category</th>
<th>Model</th>
<th>Projection Method</th>
<th>Edge Variations</th>
<th>Nodal Features?</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix Factorization</td>
<td>Laplacian Eigenmaps</td>
<td>Embedding space spanned by significant eigen vectors</td>
<td>√</td>
<td>No</td>
<td>Belkin and Niyogi</td>
</tr>
<tr>
<td></td>
<td>Graph Factorization</td>
<td>Embedding inner products approximate edge weights between nodes</td>
<td>√</td>
<td>No</td>
<td>Ahmed et al.</td>
</tr>
<tr>
<td></td>
<td>LLE</td>
<td>Embedding space spanned by significant eigen vectors</td>
<td>√</td>
<td>No</td>
<td>Hou et al.</td>
</tr>
<tr>
<td></td>
<td>HOPE</td>
<td>Factorize high-order proximity matrix</td>
<td>√</td>
<td>No</td>
<td>Ou et al.</td>
</tr>
<tr>
<td></td>
<td>GraphRep</td>
<td>Aggregate a graph's k-step representations obtained from factorization</td>
<td>√</td>
<td>No</td>
<td>Cao et al.</td>
</tr>
<tr>
<td>Random Walk-based</td>
<td>DeepWalk</td>
<td>Uniform walks; skip-gram</td>
<td>√</td>
<td>No</td>
<td>Perozzi et al.</td>
</tr>
<tr>
<td></td>
<td>Node2vec</td>
<td>Based walks; skip-gram</td>
<td>√</td>
<td>No</td>
<td>Grover and Leskovec</td>
</tr>
<tr>
<td></td>
<td>GCNE</td>
<td>Autoencoder, reconstruct adjacency matrix based on Graph Convolution</td>
<td>√</td>
<td>Yes</td>
<td>Kipf and Welling</td>
</tr>
<tr>
<td></td>
<td>GATE</td>
<td>Autoencoder, reconstruct adjacency matrix based on Graph Attention</td>
<td>√</td>
<td>Yes</td>
<td>Salehi and Darvich</td>
</tr>
<tr>
<td>Edge Recon.</td>
<td>LINE</td>
<td>Explicit 1st- and 2nd-order proximity feature extraction; negative edge sampling</td>
<td>√</td>
<td>No</td>
<td>Tang et al.</td>
</tr>
</tbody>
</table>

Table 7. Summary of Selected Prevailing Unsupervised Graph Embedding Methods

- Irrespective of method, methods should account for the severity and quantity of vulnerabilities when constructing embeddings.
  - **Value:** Support more targeted clustering of repositories.

Proposed VADW

- We design a novel Vulnerability-Associated Deep Walk (VADW) that extends the TADW's objective function to incorporate an adapted TF-IDF formulation that weights vulnerabilities based on their severity and quantity.

- The proposed VADW has two core design novelties:
  1. **Vulnerability Severity Feature Weighting:** An adapted TF-IDF formulation to account for vulnerability severity and operate on vulnerabilities within repositories
  2. **Extended TADW Objective Function:** That incorporates the Vulnerability Severity Feature Weighting mechanism during the embedding generation process.

- **Design Intuition:** VADW prioritizes low quantity high severity vulnerabilities over high quantity low severity vulnerabilities when generating embeddings.
Proposed VADW – Vulnerability Severity Feature Weighting

- In order to prioritize high severity vulnerabilities that occur with a lower frequency over low severity vulnerabilities that occur at a higher frequency, we adapt the TF-IDF function.

- TF-IDF is selected for adaptation is the vulnerability-repository relationship is analogous to term-document relationship.

- The adaptation is formally denoted as $v_{ij} = v_{fi}j \times \log \left( \frac{N}{n_{fi}} \right)$ where:
  - $v_{ij}$ is the weight of the vulnerability $i$ in the node $j$.
  - $v_{fi}j$ is the number of occurrences of vulnerability $i$ in node $j$.
  - $N$ is the total number of repositories.
  - $n_{fi}$ is the number of repositories that contain the vulnerability $i$.

Proposed VADW – Extended Objective Function

- The weights attained from the vulnerability feature weighting is stored in a matrix $V$ and integrated into the extended TADW objective function below.

\[
\min_{W,H} ||M - WTHTV^T||_F^2 + \frac{\lambda}{2} (||W||_F^2 + ||H||_F^2)
\]

**Matrix Factorization**
- $M$ is the random walk probability matrix.
- $W$ is the node embedding of each vertex.
- $H$ is the transition matrix.
- $V$ is the vulnerability feature weight matrix.
- $T$ is the feature matrix.

**Regularization Terms**
- $\frac{\lambda}{2}$ is a constant.
- $||W||_F^2$ is the regularization of the node embedding.
- $||H||_F^2$ is the regularization of the transition matrix.

**Value:** Regularization constraints normalize features to prevent scaling issues.

- Embedding generated from VADW are inputted into the k-means clustering algorithm to identify groups of repositories and users with similar vulnerability characteristics.
Presentation Overview

Project Overview: Background and Research Questions (10-12 minutes)
1. My Background
2. Cybersecurity Motivation and Existing NSF initiatives
3. Overview of GitHub cybersecurity issues
4. Research Objectives

Data Collection and Proposed Methodology (10-12 minutes)
1. Research Gaps and Questions
2. Data Collection
3. Vulnerability Assessment
4. Proposed VADW

Selected Results and Conclusion (10-12 minutes)
1. Experimental Results
2. Case Study Results
3. Conclusion

Experiments

• We evaluate VADW against three state-of-the-art unsupervised graph embedding models that account for nodal features: TADW (Yang et al., 2015), GCAE (Kipf and Welling, 2016), and GATE (Salehi and Davulcu, 2019).

• Therefore, we evaluate the proposed VADW with two sets experiments that evaluate clustering quality:
  1. All Repo Types to evaluate VADW’s ability to represent all repo types (root, forks,).
  2. Root Repositories Only To evaluate how VADW performs on repositories under the control of the scientific CI (forks may be outside the purview of a CI).
    • This evaluation also examines how a greater vulnerability variance across repositories (omitting forks greatly reduces the number of highly similar repositories) affects performance.
### Experiment 1 – All Repository Types

#### Table 8. Experiment 1 Results (*: p<0.1, **: p<0.05, ***: p<0.01; Evaluated with five commonly-used external cluster evaluation metrics)

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Method</th>
<th>Evaluation Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>ARI</td>
</tr>
<tr>
<td>CI 1</td>
<td>VADW</td>
<td>0.126</td>
</tr>
<tr>
<td></td>
<td>TADW</td>
<td>0.044***</td>
</tr>
<tr>
<td></td>
<td>GCAE</td>
<td>0.024***</td>
</tr>
<tr>
<td></td>
<td>GATE</td>
<td>0.036***</td>
</tr>
<tr>
<td>CI 2</td>
<td>VADW</td>
<td>-0.004</td>
</tr>
<tr>
<td></td>
<td>TADW</td>
<td>-0.016**</td>
</tr>
<tr>
<td></td>
<td>GCAE</td>
<td>-0.013**</td>
</tr>
<tr>
<td></td>
<td>GATE</td>
<td>-0.010</td>
</tr>
</tbody>
</table>

#### Figure 3. Experiment 1 Results

- **Key Observations:**
  - VADW outperformed embedding variants across all metrics for CI 1 and all metrics except completeness for CI 2.
  - Indicates that existing embedding methods do not capture critical vulnerable features properly.
  - Additionally, the VADW model performs well across multiple scientific CIs, indicating its ability to generalize across different scientific CIs.

### Table 9. Selected GitHub Repository Clustering Label Results

<table>
<thead>
<tr>
<th>Repository</th>
<th>True Label</th>
<th>VADW Label</th>
<th>TADW Label</th>
<th>Total Vulnerabilities</th>
<th>High Severity</th>
<th>Medium Severity</th>
<th>Low Severity</th>
<th>Secret</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>W835334/VAPOR</td>
<td>High Severity</td>
<td>High Severity</td>
<td>Low Severity</td>
<td>317</td>
<td>28</td>
<td>32</td>
<td>357</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

#### Figure 4. High Severity File Permission Vulnerability

- The results indicate that capturing weights of vulnerability severities more comprehensively represents a repository (of any type) than un-weighted features alone.

- For example, TADW did not weigh a high severity vulnerability and as a result the repository was mislabeled. However the VADW embeddings were labeled correctly.

- Accurate identification of high priority fork repositories allows CIs to contact key users within or outside of a scientific CI for vulnerability remediation.
Experiment 2 – Root Repositories Only

<table>
<thead>
<tr>
<th>Dataset Method</th>
<th>Evaluation Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>VADW</td>
<td>ARI 0.025 AMI 0.024 Completeness 0.242 Homogeneity 0.234 V-Measure 0.238</td>
</tr>
<tr>
<td>TADW</td>
<td>-0.024*** -0.008** 0.220** 0.188*** 0.203***</td>
</tr>
<tr>
<td>GCAE</td>
<td>-0.029*** -0.027*** 0.176*** 0.100*** 0.127***</td>
</tr>
<tr>
<td>GATE</td>
<td>-0.023*** -0.008*** 0.281 0.162*** 0.205***</td>
</tr>
</tbody>
</table>

Table 10. Experiment 2 Results (*:p<0.1, **: p<0.05, ***: p<0.01; Evaluated with five commonly-used external cluster evaluation metrics)

**Figure 3. Experiment 2 Results**

- **Key Observations:**
  - VADW outperformed embedding variants across all metrics except completeness for CI 1 and across all metrics for CI 2.
  - Indicates that VADW's performance over baselines improves with increased repository feature variability.

**Experiment 2 – Root Repositories Only**

<table>
<thead>
<tr>
<th>Repository</th>
<th>True Label</th>
<th>VADW Label</th>
<th>TADW Label</th>
<th>Total Vulnerabilities</th>
<th>High Severity</th>
<th>Medium Severity</th>
<th>Low Severity</th>
<th>Secret</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>NCAR/GENPRO</td>
<td>High Severity</td>
<td>High Severity</td>
<td>Low Severity</td>
<td>509</td>
<td>14</td>
<td>13</td>
<td>482</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 11. Selected GitHub Repository Clustering Label Results

- The results indicate that capturing weights of vulnerability severities more comprehensively represents a root repository than un-weighted features.

- Table 11 illustrate an example of a repository TADW mis-clustered due to a high severity, low quantity file permission vulnerability.
  - However, the VADW embedding of this repository was labeled correctly.

- Accurate identification of high priority root repositories allows CIs to immediately conduct effective remediation as root repositories are under their direct control.

**Figure 6. High Severity File Permission Vulnerability**

![Graph showing VADW against graph embedding baselines for CI 1 and CI 2](image_url)
Case Study: Clustering Similar Repositories

• To showcase the practical value of the VADW for CIs and cybersecurity professionals, we conduct a case study grouping all scientific CI 1’s repositories.

• The case study illustrates how security analysts can use the vulnerability assessment and VADW procedures for targeted repository prioritization and remediation activities.

• Key Observations:
  1. DE is a repository that contains all the public code used in CI 1’s Discovery Environment life science research web portal that provides access to the data store and compute resources of the CI.
     • The repository contains secret, password, and insecure function vulnerabilities and has been forked 10 times.
  2. The Atmosphere-guides repository contains documentation and guides for Atmosphere, an open-source cloud computing project.
     1. Contains 167 high severity insecure input, insecure functions, and file permissions C/C++ vulnerabilities.

Table 12. Selected Vulnerable Repositories Within Clusters

Cluster | Vulnerability | Severity | Repository Count
--- | --- | --- | ---
D | Password | Low | 8,096
E | Secret | High | 855
F | Insecure Function | High | 16
G | Insecure Input | High | 16
H | Insecure Function | High | 55
I | File Permission | High | 16

Table 5. Clustered Repositories

Figure 4. VADW Vulnerability Grouping Procedure

Figure 5. Clustered Repositories
Case Study: Clustering Similar Repositories

- Through the clustering process, key repositories were automatically identified for remediation.

- The mitigation strategies developed for these repositories can guide the CI in how to remediate similar vulnerabilities for entire repository clusters.

<table>
<thead>
<tr>
<th>Vulnerability</th>
<th>Description</th>
<th>Example</th>
<th>Potential Threat</th>
<th>Remediation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secret</td>
<td>A potential password/key</td>
<td>AWS API Key</td>
<td>Abused cloud computing resources</td>
<td>Discontinue use of all leaked secrets</td>
</tr>
<tr>
<td>File permission</td>
<td>File may have dangerous permissions</td>
<td>chmod(filePath, 0664)</td>
<td>Allow malicious file to corrupt file system</td>
<td>Patch vulnerable code snippets and alert repository users to update</td>
</tr>
<tr>
<td>Insecure function</td>
<td>Function can be vulnerable</td>
<td>use of insecure and deprecated function (mktemp)</td>
<td>Allow attacker to modify file before it is opened</td>
<td>Patch vulnerable code snippets and alert repository users to update</td>
</tr>
<tr>
<td>Insecure input</td>
<td>Dangerous handling of user input</td>
<td>use of unsafe yaml_load. Allows instantiation of arbitrary objects.</td>
<td>Malicious remote code execution</td>
<td>Patch vulnerable code snippets and alert repository users to update</td>
</tr>
</tbody>
</table>

Table 13. Remediation Strategies for Selected High Severity Vulnerabilities

Conclusion and Future Directions

- Scientific CI utilize SCRs such as GitHub to facilitate the development of open-source software for advanced scientific processes.

- However, these repositories often contain vulnerabilities that require vulnerability scanners capable of scanning source code directly to identify.
  - These vulnerabilities can expose scientific CIs to cyber attacks that can disrupt scientific research.

- In this study, we develop a novel Vulnerability-Associated Deep Walk (VADW) graph embedding approach to automatically identify and cluster repositories with similar vulnerabilities.
Conclusion and Future Directions

• The value of the VADW was illustrated with a case study of grouping repositories from a major scientific CI with similar vulnerabilities.

• Targeted mitigation and remediation strategies were proposed to help CIs address vulnerabilities within selected clusters.

• There are several promising directions for future research:
  1. Scan repositories for software vulnerabilities at a commit level to identify when vulnerabilities were introduced to the repositories.
  2. Identify how vulnerabilities propagate across repositories.
  3. Pinpoint key users posting vulnerabilities for targeted security awareness trainings.
  4. Group emerging categories of scientific inquiry (e.g., GitHub repos from AI conferences).

Thank you!

Questions or Comments?

—
Sagar Samtani, Ph.D.
Kelley School of Business, Indiana University
ssamtani@iu.edu
www.sagarsamtani.com (access to my papers)